Объектно-ориентированное проектирование ЭИС

Структурная декомпозиция ЭИС на основе объектно-ориентированного подхода отличается от функционально-ориентированного подхода лучшей способностью отражать динамическое поведение системы в зависимости от возникающих событий. Модель проблемной области рассматривается как совокупность взаимодействующих во времени объектов. при этом статическая структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами. Каждый объект системы обладает своим собственным поведением, моделирующим поведение объекта реального мира. Модель предметной области рассматривается как совокупность взаимодействующих объектов во времени. Конкретные процессы обработки данных формулируются в виде последовательности взаимодействия объектов.

Конечным результатом процесса объектно-ориентированного проектирования должно стать множество классов объектов с присоединенными методами обработки атрибутов. Объектно-ориентированный подход предполагает совместное моделирование данных и процессов. При этом модели проблемной области в репозитории постепенно уточняются.

Концептуальной основой объектно-ориентированного подхода является объектная модель. Основными ее элементами являются: **абстрагирование, инкапсуляция, модульность, иерархия, типизация, параллелизм, устойчивость, полиморфизм, наследование**.

Система объектно-ориентированных моделей последовательно разворачивается по направлению от модели общего представления функциональности ЭИС к модели дина­мического взаимодействия объектов, на основе которой могут быть сгенерированы классы объектов в конкретной программ­но-технической среде.

В настоящее время для объектно-ориентированного модели­рования проблемной области широко используется унифициро­ванный язык моделирования UML (Unified Modeling Language), который фактически является стан­дартом по объектно-ориентированным технологиям:

* не зависит от объектно-ориентированных (ОО) языков программирования;
* не зависит от используемой методологии разработки проекта;
* может поддерживать любой ОО язык программирования.

# Оcновные понятия UML.

### Сущности в UML

В UML определены четыре типа сущностей: **структурные, поведенческие, группирующие и аннотационные**. Сущности являются основными объектно-ориентированными элементами языка, с помощью которых создаются модели.

**Структурные сущности** - это имена существительные в моделях на языке UML. Как правило, они представляют статические части модели, соответствующие концептуальным или физическим элементам системы. Примерами структурных сущностей являются "класс", "интерфейс", "кооперация", "прецедент", "компонент", "узел", "актор".

**Поведенческие сущности** являются динамическими составляющими модели UML. Это глаголы, которые описывают поведение модели во времени и в пространстве. Существует два основных типа поведенческих сущностей:

* взаимодействие - это поведение, суть которого заключается в обмене сообщениями между объектами в рамках конкретного контекста для достижения определенной цели;
* автомат - алгоритм поведения, определяющий последовательность состояний, через которые объект или взаимодействие проходят в ответ на различные события.

**Группирующие сущности** являются организующими частями модели UML. Это блоки, на которые можно разложить модель. Такая первичная сущность имеется в единственном экземпляре - это пакет.

Пакеты представляют собой универсальный механизм организации элементов в группы. В пакет можно поместить структурные, поведенческие и другие группирующие сущности. В отличие от компонентов, которые реально существуют во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только в процессе разработки.

**Аннотационные сущности** - это пояснительные части модели UML: комментарии для дополнительного описания, разъяснения или замечания к любому элементу модели. Имеется только один базовый тип аннотационных элементов - примечание. Примечание используют, чтобы снабдить диаграммы комментариями или ограничениями, выраженными в виде неформального или формального текста.

### Отношения в UML

В языке UML определены следующие типы отношений: **зависимость, ассоциация, обобщение и реализация**. Эти отношения являются основными связующими конструкциями UML и также как сущности применяются для построения моделей.

**Зависимость (dependency)** - это семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, может повлиять на семантику другой, зависимой.

**Ассоциация (association)** - структурное отношение, описывающее совокупность смысловых или логических связей между объектами.

**Обобщение (generalization)** - это отношение, при котором объект специализированного элемента (потомок) может быть подставлен вместо объекта обобщенного элемента (предка). При этом, в соответствии с принципами объектно-ориентированного программирования, потомок (child) наследует структуру и поведение своего предка (parent).

**Реализация (realization)** является семантическим отношением между классификаторами, при котором один классификатор определяет обязательство, а другой гарантирует его выполнение. Отношение реализации встречаются в двух случаях:

* между интерфейсами и реализующими их классами или компонентами;
* между прецедентами и реализующими их кооперациями.

### Общие механизмы UML

Для точного описания системы в UML используются, так называемые, общие механизмы:

* **спецификации** (specifications) - текстовое представление соответствующей конструкции языка. Графическая нотация UML используются для визуализации системы, а с помощью спецификаций описывают ее детали.;
* **дополнения** (adornments) – позволяет визуализировать некоторые детали спецификаций (видимость атрибутов/операций, комментарии);
* **деления** (common divisions):
  + на классы и объекты: класс - это абстракция, а объект - конкретное воплощение этой абстракции. В связи с этим, практически все конструкции языка характеризуются двойственностью "класс/объект", также прецеденты/экземпляры прецедентов, компоненты/экземпляры компонентов, узлы/экземпляры узлов.
  + на интерфейс и его реализацию. Интерфейс декларирует обязательства, а реализация представляет конкретное воплощение этих обязательств и обеспечивает точное следование объявленной семантике. В связи с этим, почти все конструкции UML характеризуются двойственностью "интерфейс/реализация". Например, прецеденты реализуются кооперациями, а операции - методами.
* **расширения** (extensibility mechanisms) - UML является открытым языком, то есть допускает контролируемые расширения, чтобы отразить особенности моделей предметных областей.

Механизмы расширения UML включают:

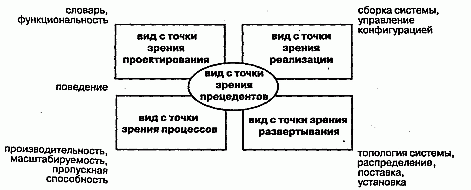
* стереотипы (stereotype) - расширяют словарь UML, позволяя на основе существующих элементов языка создавать новые, ориентированные для решения конкретной проблемы;
* помеченные значения (tagged value) - расширяют свойства основных конструкций UML, позволяя включать дополнительную информацию в спецификацию элемента;
* ограничения (constraints) - расширяют семантику конструкций UML, позволяя создавать новые и отменять существующие правила.

Совместно эти три механизма расширения языка позволяют модифицировать его в соответствии с потребностями проекта или особенностями технологии разработки.

## Точка зрения в объектно-ориентированном моделировании.

Если в методологии функционального моделирования SADT точка зрения определяется лишь как возможность существования нескольких моделей одной системы, то в объектно-ориентированном подходе выделенные точки зрения имеют вполне определенное назначение и заостряют внимание на определенном аспекте функционирования системы.

Точки зрения на архитектуру ИС:



*Вид с точки зрения прецедентов* (Use case view) охватывает прецеденты, которые описывают поведение системы, наблюдаемое конечными пользователями, аналитиками и тестировщиками. В языке UML статические аспекты этого вида передаются диаграммами прецедентов, а динамические - диаграммами взаимодействия, состояний и действий.

*Вид с точки зрения проектирования* (Design view) охватывает классы, интерфейсы и кооперации, формирующие словарь задачи и ее решения. Этот вид поддерживает прежде всего функциональные требования, предъявляемые к системе, то есть те услуги, которые она должна предоставлять конечным пользователям. С помощью языка UML статические аспекты этого вида можно передавать диаграммами классов и объектов, а динамические - диаграммами взаимодействия, состояний и действий.

*Вид с точки зрения процессов* (Process view) охватывает нити и процессы, формирующие механизмы параллелизма и синхронизации в системе. Этот вид описывает главным образом производительность, масштабируемость и пропускную способность системы. В UML его статические и динамические аспекты визуализируются теми же диаграммами, что и для вида с точки зрения проектирования, но особое внимание при этом уделяется активным классам, которые представляют соответствующие нити и процессы.

*Вид с точки зрения реализации* (Implementation view) охватывает компоненты и файлы, используемые для сборки и выпуска конечного программного продукта. Этот вид предназначен в первую очередь для управления конфигурацией версий системы, составляемых из независимых (до некоторой степени) компонентов и файлов, которые могут по-разному объединяться между собой. В языке UML статические аспекты этого вида передают с помощью диаграмм компонентов, а динамические - с помощью диаграмм взаимодействия, состояний и действий.

*Вид с точки зрения развертывания* (Deployment view) охватывает узлы, формирующие топологию аппаратных средств системы, на которой она выполняется. В первую очередь он связан с распределением, поставкой и установкой частей, составляющих физическую систему. Его статические аспекты описываются диаграммами развертывания, а динамические - диаграммами взаимодействия, состояний и действий.

Каждый из перечисленных видов может считаться вполне самостоятельным, но они взаимодействуют друг с другом. Например, узлы вида с точки зрения развертывания содержат компоненты, описанные для вида с точки зрения реализации, а те, в свою очередь, представляют собой физическое воплощение классов, интерфейсов, коопераций и активных классов из видов с точки зрения проектирования и процессов. UML позволяет отобразить каждый из пяти перечисленных видов и их взаимодействия.

## Диаграммы UML

Система объектно-ориентированных моделей в соответствии с нотациями UML включает в себя следующие диаграммы:

* диаграмму прецедентов использования (Use-case diagram);
* диаграмму классов объектов (Class diagram);
* диаграммы состояний (Statechart diagram);
* диаграммы взаимодействия объектов (Interaction diagram);
* диаграммы деятельностей (Activity diagram);
* диаграммы пакетов (Package diagram);
* диаграмму компонентов (Component diagram);
* диаграмму размещения (Deployment diagram).

**Диаграмма прецедентов использования** (Use-case diagram) выявляет основные бизнес-процессы как последовательности транзакций, которые должны выполняться целиком, когда выполнение обособленно­го подмножества действий не имеет значения без выполнения всей последовательности. Прецеденты использования инициируются из внешней среды пользователями ЭИС, называемыми актера­ми. На этом уровне моделирования не раскрывается механизм реализации процессов.

Актер (актор) инициирует выполнение прецедента использования и получает от него результаты. заимодействие (ассоциация) актера с прецедентом использования осуществляется в результате со­бытия. Один актер может участвовать в нескольких прецедентах использования, а в одном прецеденте использования может быть занято несколько актеров.



В реализации прецедента использования возможно выделение нескольких потоков событий:

* основной поток событий, который приводит к требуемому результату наиболее коротким путем, например выполнение заказа без задержек;
* альтернативные потоки событий, например временное откладывание или полный отказ от выполнения заказов. Основной и альтернативный потоки событий в модели прецедентов использования описываются в виде неформальных текстовых комментариев.

Несколько прецедентов использования может иметь общую часть, выделяемую в самостоятельный прецедент использования, с которым устанавливаются отношения вкючения/использования (**include/use**). С другой стороны, некоторые прецеденты использования могут быть расширены деталями. В таком случае создается дополнительный прецедент использования, с которым устанавливаются отношения расширения (**extend**).



Разработка диаграммы преследует следующие цели:

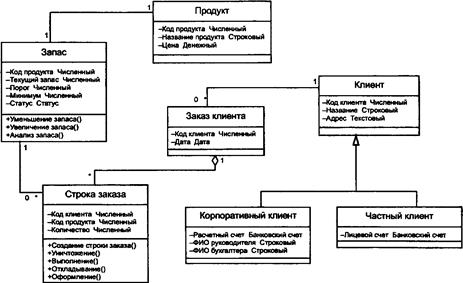
* определить общие границы и контекст моделируемой предметной области;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

**Диаграммы классов объектов** (Class diagram) отображают статическую структуру классов объектов аналогично ER-диаграмме функционально-ориентированно­го подхода. Эта диаграмма рассматривает внутреннюю структуру проблемной области, иерархию классов объектов, статические связи объектов.

Классы объектов могут иметь различные стереотипы поведения: объекты-сущности, управляющие объекты, интерфейсные объекты.

Объекты, отражаемые в диаграмме классов объектов, связываются статическими отношениями, которые отражают постоянные связи между объектами независимо от выполнения конкретного бизнес – процесса:

* Зависимость
* Обобщение
* Ассоциация



*Фрагмент диаграммы классов*

**Диаграмма состояний** (Statechart diagram) отображает поведение объектов одного класса в динамике, связь состояний объектов с событиями и определяет:

* какие типичные состояния проходит объект;
* какие события ведут к изменению состояния объекта;
* какие действия объект выполняет, когда он получает сообщение об изменении состояния;
* как объекты создаются и уничтожаются (входные и выходные точки диаграммы).

В диаграмме состояний используются следующие понятия:

*Входная точка* определяет событие, которое образует начальное состояние объекта. В точку входа нельзя перейти из состояния объекта.

*Выходная точка* определяет завершение существования объекта. Из точки выхода нет перехода состояния.

*Состояние* представляет ситуацию, в течение которой выполняется непрерывная деятельность или объект находится в стационарном положении. Состояние определяется как набор значений атрибутов и отношений, связанных с объектом. С каждым состоянием связано одно событие или более, которые могут его изменить. Для состояния задаются имена всех связанных с ним переходов в другие состояния.

*Переход состояний* определяет изменение в состоянии объекта, которое происходит в результате события, возникшего в то время, когда объект находился в данном состоянии. Каждый переход состояний должен иметь уникальное имя.

Переход состояний описывается следующими атрибутами:

* Назначение- состояние объекта, в которое перейдет объект после перехода состояния.
* Вызов- имя события, которое вызывает переход состояний. Имена событий должны быть идентичными в определении класса и состояния. Вызываемые события могут быть либо внешними, осуществляемыми актерами, либо внутренними, связанными с поведением других объектов, либо временными, связанными с истечением заданного интервала времени.
* Условие перехода *-* это логическое выражение, связанное с атрибутами объекта, которое должно быть проверено для выбора перехода состояния. Условие перехода задается в том случае, если происходит событие, в результате которого может произойти неоднозначный переход состояний. Условия переходов для одного исходного состояния должны быть взаимоисключающими.
* Действие - атрибут, информационно описывающий сущность действия, которое должно выполняться при переходе состояний. Этому действию будет соответствовать некоторая процедура, реализующая метод класса объектов.

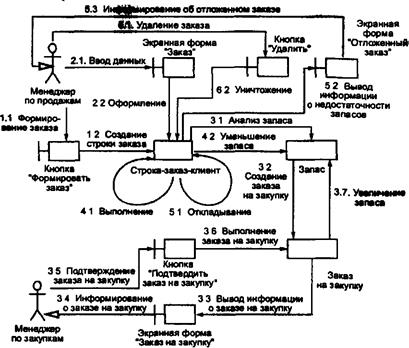


*Диаграммы состояний для объекта «строка заказа»*

**Диаграмма взаимодействия объектов** (Interaction diagram) отображает динамическое взаимодействие объектов в рамках одного прецедента использования, имеет два варианта:

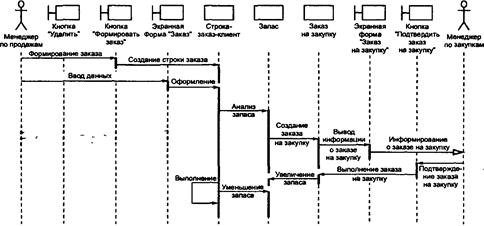
* диаграмма последовательностей
* кооперативная диаграммы:

**Диаграмма последовательности** (Sequence diagram) моделирует взаимодействия объектов во времени, используется для точного определения логики сценария выполнения прецедента. Диаграммы последовательностей отображают типы объектов, взаимодействующих при исполнении прецедентов, сообщения, которые они посылают друг другу, и любые возвращаемые значения, ассоциированные с этими сообщениями



*Диаграмма последовательностей для прецедента Выполнение заказа*

**Диаграмма кооперации** (collaboration diagram) представляет не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии. Этот тип диаграмм позволяет описать взаимодействия объектов, абстрагируясь от последовательности передачи сообщений. На этом типе диаграмм в компактном виде отражаются все принимаемые и передаваемые сообщения конкретного объекта и типы этих сообщений.



*Диаграмма кооперативного поведения для основного потока событий   
прецедента использования Выполнить заказ*

Понятие **кооперации** (collaboration) - фундаментальное понятие UML - обозначает множество взаимодействующих с определенной целью объектов в общем контексте моделируемой системы.

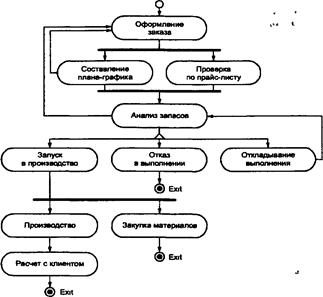
Кооперация может быть представлена на двух уровнях:

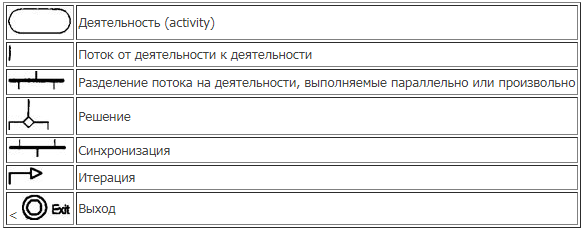
* уровне спецификации - показывает роли классификаторов и роли ассоциаций в рассматриваемом взаимодействии;
* уровне примеров - указывает экземпляры и связи, образующие отдельные роли в кооперации.

**Диаграммы деятельностей** (Activity diagram) моделируют процесс выполнения операций, являются частным случаем диаграмм состояний, позволяют реализовать в языке UML особенности процедурного и синхронного управления.

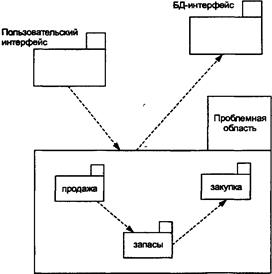
иаграммы взаимодействий не отражают детально порядок выполнения операций в части разветвлений, циклических повторений, параллельности/произвольности действий. Диаграмма деятельностей исправляет данные недостатки. Под деятельностью будем понимать некоторую работу, которая может быть декомпозирована на совокупность действий.

Диаграммы деятельности позволяют получить полную картину поведения системы и легко оценивать влияние изменений в отдельных вариантах использования на конечное поведение системы. Любая деятельность может быть подвергнута дальнейшей декомпозиции и представлена в виде отдельной диаграммы деятельности или спецификации (словесного описания).





**Диаграмма пакетов** (Package diagram) содержит пакеты классов и зависимости между ними. Зависимость между двумя пакетами имеет место в том случае, если изменения в определении одного элемента влекут за собой изменения в другом. По отношению к пакетам можно использовать механизм обобщения.



В объектно-ориентированном подходе пакет содержит множество взаимосвязанных классов объектов и соответствует понятию «подсистема функционально-ориентированного подхода». Один прецедент использования может требовать классы объектов из разных пакетов. Класс объектов обычно назначается одному пакету, но с позиции достижения разных подцелей может входить в состав разных пакетов.

Пакетная технология группирования классов объектов позволяет упростить:

* разработку и эксплуатацию ЭИС;
* гибкую адаптацию типовых компонентов с позиции их повторного использования;
* оптимизацию клиент-серверной архитектуры ЭИС.

Обычно ЭИС разбивается на: функциональные и обеспечивающие пакеты.

Функциональные пакеты, соответствующие решаемым проблемам (задачам), объединяются в один общий пакет «Проблемная область». Каждый пакет, в свою очередь, может быть разбит на подпакеты в соответствии с семантической близостью и теснотой взаимодействия классов объектов.

Обычно пакеты проблемной области содержат иерархии обобщения и агрегации. Классы объектов, требуемые в нескольких подсистемах, выделяются в самостоятельные пакеты. В одном пакете, как правило, определяется не более 20 компонентов, обычно 5–15.

С обеспечивающей точки зрения ЭИС разбивают на пять основных пакетов:

* «Интерфейс», объекты которого реализуют функции взаимодействия пользователей с ЭИС по вводу-выводу информации и обмен сообщениями между подсистемами;
* «База данных», объекты которого выполняют доступ к данным во внешней памяти;
* «Управление задачами», объекты которого осуществляют функции диспетчеризации и маршрутизации обработки объектов, например, в системе управления рабочими потоками;
* «Утилиты», объекты которого осуществляют вспомогательные функции, например преобразование форматов данных;
* Обеспечивающие пакеты, т.е. работающие по принципу «клиент-серверной» архитектуры, выполняющие серверные функции для функциональных объектов-клиентов.

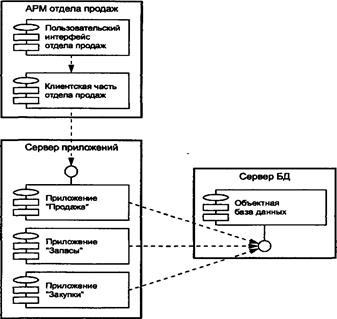
**Диаграммы компонентов и размещения**

Диаграмма компонентов (Component diagram) отображает зависимости программных компонентов, которые представляются в виде исходных, откомпилированных и исполняемых программных кодов объектов. Один компонент, как правило, соответствует программному коду одного пакета классов объектов.

Компонент в своем составе имеет интерфейсный класс объектов, через который осуществляется доступ к остальным классам объектов компонента. С помощью интерфейса объекты других компонентов обращаются не к конкретным объектам рассматриваемого компонента, а к его интерфейсному объекту. Таким образом, упрощается взаимодействие компонентов между собой, когда при доступе к компоненту из других компонентов не требуется знать внутреннюю структуру этого компонента. Компонент, к которому осуществляется обращение, может быть необъектно-ориентированным.

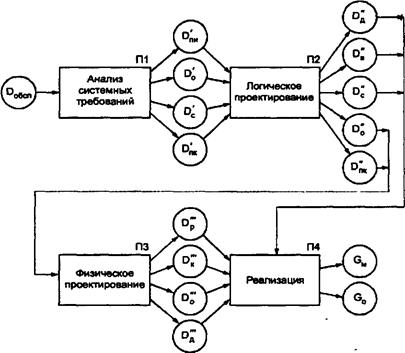
Достаточно, чтобы у такого компонента был только один интерфейсный класс объектов, который транслирует запросы к компоненту в вызовы обычных процедур. У компонентов может быть несколько интерфейсов.

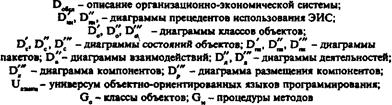
**Диаграмма размещения** (Deployment diagram) отображает топологию расположения компонентов по узлам вычислительной сети. Отдельный компонент всегда располагается на одном компьютере-сервере. На одном компьютере-сервере может располагаться несколько компонентов.



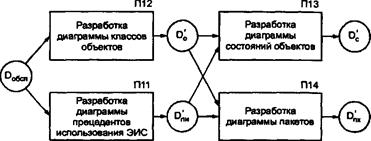
# Технологическая сеть проектирования ЭИС на основе использования объектно-ориентированной CASE-технологии

Рассмотрим технологическую сеть проектирования ЭИС на основе использования объектно-ориентированной CASE-технологии, для которой характерны последовательное расширение и уточнение моделей на различных стадиях жизненного цикла ЭИС анализа системных требований, логического и физического проектирования, реализации.





Технологическая сеть анализа системных требований к ЭИС представлена на рис:



Doбсл- описание организационно-экономической системы;

D'пи - диаграмма прецедентов использования ЭИС,

D'о - диаграмма классов объектов;

D’c - диаграммы состояний объектов,

D'пк - диаграмма пакетов.

На входе этапа анализа системных требований используется описание организационно-экономической системы (Dобсл), полученное в ходе работ по анализу и проектированию бизнес-процессов. Эти материалы содержат описание организационной структуры, структуры материальных, финансовых и информационных потоков, которое может быть выполнено либо с помощью традиционных средств графического отображения, либо с помощью определенных методологий бизнес-реинжиниринга, например с помощью той же объектно-ориентированной методологии.

Так, в объектно-ориентированной методологии анализа и проектирования бизнес-процессов предусматриваются:

* + 1. Описание бизнес-процессов как прецедентов использования, актерами которых служат внешние участники бизнес-процессов (клиенты, поставщики, субподрядчики, инвесторы, финансовые компании, государственные органы).
    2. Задание порядка разработки и автоматизации бизнес-процессов в соответствии с определенными критериями, например наибольшим эффектом для заказчика, простотой и быстротой разработки и т. д.
    3. Неформальное словесное описание бизнес-процессов.

Структура основных бизнес-объектов и их взаимодействии описывается в соответствии с требованиями модели классов объектов.

Анализ системных требований начинается с идентификации основных прецедентов использования (D'пи) и объектов-сущностей (D'o), которые будут применяться в информационной системе. Работы по идентификации прецедентов использования и классов объектов-сущностей, как правило, выполняются параллельно. В случае объектно-ориентированного оформления результатов предпроектного обследования данная работа упрощается в силу однозначности соответствия бизнес-процессов и прецедентов использования ЭИС, бизнес-объектов и объектов-сущностей.

Разработка D'пи -диаграммы прецедентов использования ЭИС (преобразователь ПИ) предполагает выделение тех последовательностей транзакций, которые будут автоматизировать требуемые бизнес-процессы. При этом определяются основные пользователи-актеры, взаимодействующие с прецедентами использования.

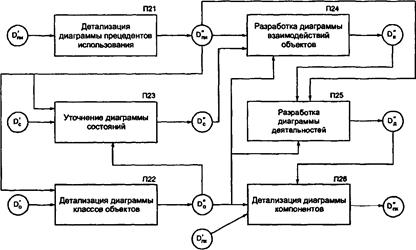
Разработка D'o - диаграммы классов объектов (преобразователь П12) предполагает задание состава основных атрибутов и определение характера взаимосвязей классов объектов.

Разработка D'с - диаграммы состояний объектов (преобразователь П13) осуществляется только для классов объектов со сложным поведением. При этом рассматриваются все прецеденты использования, в которых объекты данного класса используются и меняют свои состояния.

Разработка D'пк - диаграммы пакетов (преобразователь П14) осуществляется путем группировки классов объектов по подсистемам. На этапе анализа системных требований определяется состав пакетов, относящихся к пакету «Проблемная область». При этом выделяются функциональные пакеты, которые объединяют классы объектов, реализующие функции управления, и базовые пакеты с нормативно-справочной информацией, общие для функциональных пакетов.

## Логическое проектирование

На этапе логического проектирования ЭИС осуществляются детализация моделей прецедентов использования, классов объектов, состояний, пакетов и разработка моделей взаимодействия объектов и деятельностей, которые определяют характер методов (процедур) обработки объектов:



Детализация D''пи - диаграммы прецедентов использования (преобразователь П21) предполагает разработку основных и альтернативных потоков событий, которые могут быть представлены самостоятельными диаграммами прецедентов использования. Кроме того, могут быть выделены прецеденты использования, расширяющие набор функций основных прецедентов, или из нескольких прецедентов использования могут быть выделены общие функции в самостоятельные прецеденты. При этом соответственно задаются отношения расширения и использования.

Детализация D''о - диаграммы классов объектов (преобразователь П22) выполняется путем уточнения классов объектов-сущностей и введения интерфейсных и управляющих классов объектов. Интерфейсные классы объектов соответствуют актерам прецедентов использования, а управляющие классы объектов - координирующим функциям обработки объектов-сущностей.

Уточнение D"с- диаграммы состояний объектов (преобразователь П23) выполняется в связи с детализацией диаграммы прецедентов использования D''пи и диаграммы классов объектов D''o.

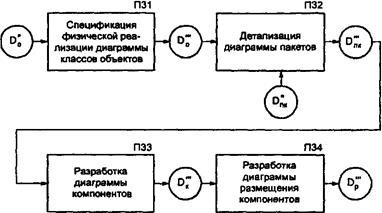
Разработка D''в - диаграммы взаимодействий (преобразователь П24) выполняется для каждого прецедента использования с учетом построенных диаграмм классов объектов и состояний. В частности, сообщение от одного объекта к другому в диаграмме взаимодействия должно соответствовать событию, вызывающему переход состояния объекта получателя сообщения в диаграмме состояний. Аналогично внешнее событие в диаграмме взаимодействий, вызываемое актером, соответствует событию, осуществляющему переход состояния объекта в диаграмме состояний.

Разработка D''д - диаграммы деятельностей (преобразователь П25) уточняет характер взаимодействия объектов не в одном, а в нескольких прецедентах использования. Если диаграммы взаимодействий объектов формируют набор методов обработки объектов, то диаграммы деятельностей дают спецификацию алгоритмов для последующего программирования процедур этих методов.

Детализация D''пк-диаграммы пакетов (преобразователь П26) связана с уточнением состава классов объектов-сущностей и появлением интерфейсных и управляющих классов объектов. Например, интерфейсные и управляющие классы объектов могут быть выделены в самостоятельные обеспечивающие пакеты.

## Физическое проектирование

На этапе физического проектирования происходит детализация диаграмм классов объектов и пакетов с позиции их реализации в конкретной программно-технической среде:



D"'o - диаграммы классов объектов,

D"'с - диаграммы состояний объектов;

D"'пк -диаграммы пакетов;

D"'к - диаграмма компонентов,

D"'р - диаграмма размещения компонентов.

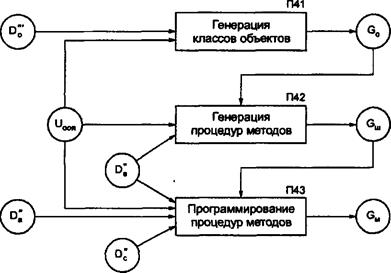
Спецификация физической реализации D"'o - диаграммы классов объектов (преобразователь П31) предусматривает определение форматов данных для атрибутов и методов реализации отношений (ключей, указателей, процедур) классов объектов

Детализация D"'пк - диаграммы пакетов (преобразователь П32) предполагает разработку обеспечивающих компонентов: базы данных, управления задачами, вспомогательных функций.

Разработка D"'к - диаграммы компонентов (преобразователь ПЗЗ) и D"'р - диаграммы размещения компонентов (преобразователь П34) реализует клиент-серверную технологию и определяет схему размещения компонентов по узлам вычислительной сети.

## Реализация ЭИС

На этапе реализации ЭИС осуществляются кодогенерация классов объектов, программирование процедур методов классов объектов, наполнение баз данных и размещение компонентов по узлам вычислительной сети:



Uооя, - универсум объектно-ориентированных языков программирования;

D"'o - диаграмма классов объектов;

D"'c- диаграммы состояний объектов;

D"'пк - диаграмма пакетов; D"в - диаграммы взаимодействий;

D"а- диаграмма активностей;

D"'к - диаграмма компонентов;

D"р- диаграмма размещения компонентов;

Gо - классы объектов;

Gш - шаблоны процедур методов класса объектов;

Gм - процедуры методов.

Генерация Go - классов объектов (преобразователь П41) в конкретной объектно-ориентированной программной среде (C++, Visual Basic, Pascal и т.д.), выбираемой из Uооя - универсума объектно-ориентированных языков программирования, осуществляется на основе диаграммы классов объектов D"'o .

Генерация Gш - шаблонов процедур методов класса объектов (преобразователь П42) в конкретной объектно-ориентированной программной среде (C++, Visual Basic, Pascal и т.д.), выбираемой из универсума объектно-ориентированных языков программирования, производится на основе диаграммы взаимодействий объектов D"в.

Программирование Gм процедур методов класса объектов (преобразователь П43) с помощью объектно-ориентированного языка программирования выполняется на основе Dш - шаблонов процедур методов классов объектов по спецификациям D"д - диаграмм деятельностей и D"с - состояний объектов.